Langara College
CPSC 2150
Assignment #7: Hashing

Assignment due with Brightspace at 11:50pm on March 28

Read chapter 10 of the textbook by Drozdek
§10.4.1, §10.4.2, §10.5.1, §10.6 not covered in the course
Goodrich

Purpose

The purpose of this assignment is to compare linear probing against quadratic probing.
We say that quadratic probing is better than linear probing. Let's see if this is true empirically.

Implementation

Create a function `getKey()` that returns a string of random upper case letters. Pick the number of letters in the key empirically.

Run your experiments for load factors of 0.2, 0.5 and 0.8.

In the explanation here we are using a load factor of 0.5 only.

For the hash table T we will use an array of size m = 1999 strings.

Use the empty string to indicate that slot T[i] is empty (define it as a named constant).

For the hash function h(x) I suggest using the hash function presented in lecture provided in hash_function.cpp and hash_function.h

Now write two functions (do add more parameters as needed)

search(x, T) to test if a given key x is in the hash table T
insert(x, T) to insert a given key x into the hash table T

You will need two versions of each function, one that uses linear probing and the other which uses quadratic probing, so four functions in total.[1]

[1] You could pass the probing function as an argument. You could also pass an argument to distinguish linear probing from quadratic probing: if you do that, document your code properly and do not complicate your code unnecessarily. In such a case, you will not have two versions of each function.

Now perform the following experiment. Using `getKey()`[2], generate 1000 distinct keys and insert them into the hash table T. If a key generated is a repeat, ignore it and produce another key. So, keep producing and inserting new keys until T has 1000 distinct strings and the hash table is half full (OK, not perfectly half full but it should be close enough: here the load factor is 0.5). You can do this by first searching in T for the key and if it is not already there, insert it.

First do this using linear probing, then do this using quadratic probing. So you will need two hash tables, TL and TQ for linear probing and for quadratic probing[3]

Now, suppose we have a key x. We want to determine the average number of comparisons done when searching for x in the hash table T for the two cases,

(i) when x is in T
(ii) when x is not in T

And we want to do this for linear probing and quadratic probing.
So there will be four averages.

To do this for case (i), you could go through each entry that is stored in the array T, and you could search in T for this key counting the number of comparisons made by the search function, then taking the average.
I leave it to you to figure out how to do this for case (ii).

That is **one** experiment. If you repeat this 100 times, each time with 1000 different random keys, so that the load factor is α=0.5 each time, and if you take the average over the 100 experiments, you should be close to the theoretical values obtained from the formulae given in the next page.

For the quadratic probing, it's up to you whether you want to alternate between adding a square number and subtracting a square number as shown in the textbook by Drozdek on pages 553 to 554 or whether you just want to add square numbers when composing your probing sequence.

[2] You could make `getKey` a member function of a class that generates random strings.

[3] you can reuse the table when running your experiments, it's just easier to say that there is two hash tables in this explanation

Formulae

The formulae come from Donald Knuth from his 1998 edition of his book *The Art of Computer Programming*, Vol. 3, Reading, MA: Addison-Wesley and copied from your textbook by A. Drozdek, *Data Structures and Algorithms in C++*, fourth edition, 2013, Cengage Learning.

For **quadratic probing**, with a load factor α, the approximate average number of comparisons that a search requires for a successful search is

$$1 - \ln(1 - \alpha) - \frac{\alpha}{2}$$

For **quadratic probing**, with a load factor α, the approximate average number of comparisons that a search requires for an unsuccessful search is

$$\frac{1}{1 - \alpha} - \alpha - \ln(1 - \alpha)$$

For **linear probing**, with a load factor α, the approximate average number of comparisons that a search requires for a successful search is

$$\frac{1}{2}\left(1 + \frac{1}{(1 - \alpha)}\right)$$

For **linear probing**, with a load factor α, the approximate average number of comparisons that a search requires for an unsuccessful search is

$$\frac{1}{2}\left(1 + \frac{1}{(1 - \alpha)^2}\right)$$

Results

Use a table of 1999 elements.

The theoretical values come from the formulae.

Print your results.
For load factors of 0.2, 0.5 and 0.8 your program should calculate

- for linear probing
  - theoretical value when key is in the table
  - average number of comparisons done when key is in the table
  - difference[4] between theoretical value and empirical value
  - theoretical value when key is NOT in the table
  - average number of comparisons done when key is NOT in the table
  - difference between theoretical value and empirical value
- for quadratic probing
  - theoretical value when key is in the table
  - average number of comparisons done when key is in the table
  - difference between theoretical value and empirical value
  - theoretical value when key is NOT in the table
  - average number of comparisons done when key is NOT in the table
  - difference between theoretical value and empirical value

[4] use the absolute value

To submit as Assignment Bonus as a single compressed zip file

i. the source code
we have provided a directory with the files hash_function.cpp, hash_function.h (leave as is) and experiments.cpp with a Makefile
Add other files if you want to but add them to the Makefile as well.
You do not have to make a class.

ii. the results produced by your program (you could paste the results into the README)

iii. a README.txt or README.docx
   a. where you summarize your results and compare linear probing with quadratic probing – come up with some conclusions
   b. where you discuss the values that you obtained from these experiments versus the theoretical values from the formulae and an explanation of the possible discrepancies
   c. anything you want to convey to the person marking the assignment e.g. parts that you did not do

Bonus (30%)

Implement an experiment like the one above for **double hashing**.

You will need to worry about coming up with a second hashing function.
Your textbook has in Figure 10.3 on page 557 the formulae for **double hashing** for a successful search and for an unsuccessful search. The Figure 10.3 by Drozdek is also on Brightspace.

You will need to submit code, results, and some conclusions as well.

You can incorporate the double hashing in the same program.